内容に関するご質問は
kawai@cc.u-tokyo.ac.jp
まで、お願いします。

第193回　お試しアカウント付き
並列プログラミング講習会
「ライブラリ利用：科学技術計算の効率化入門」

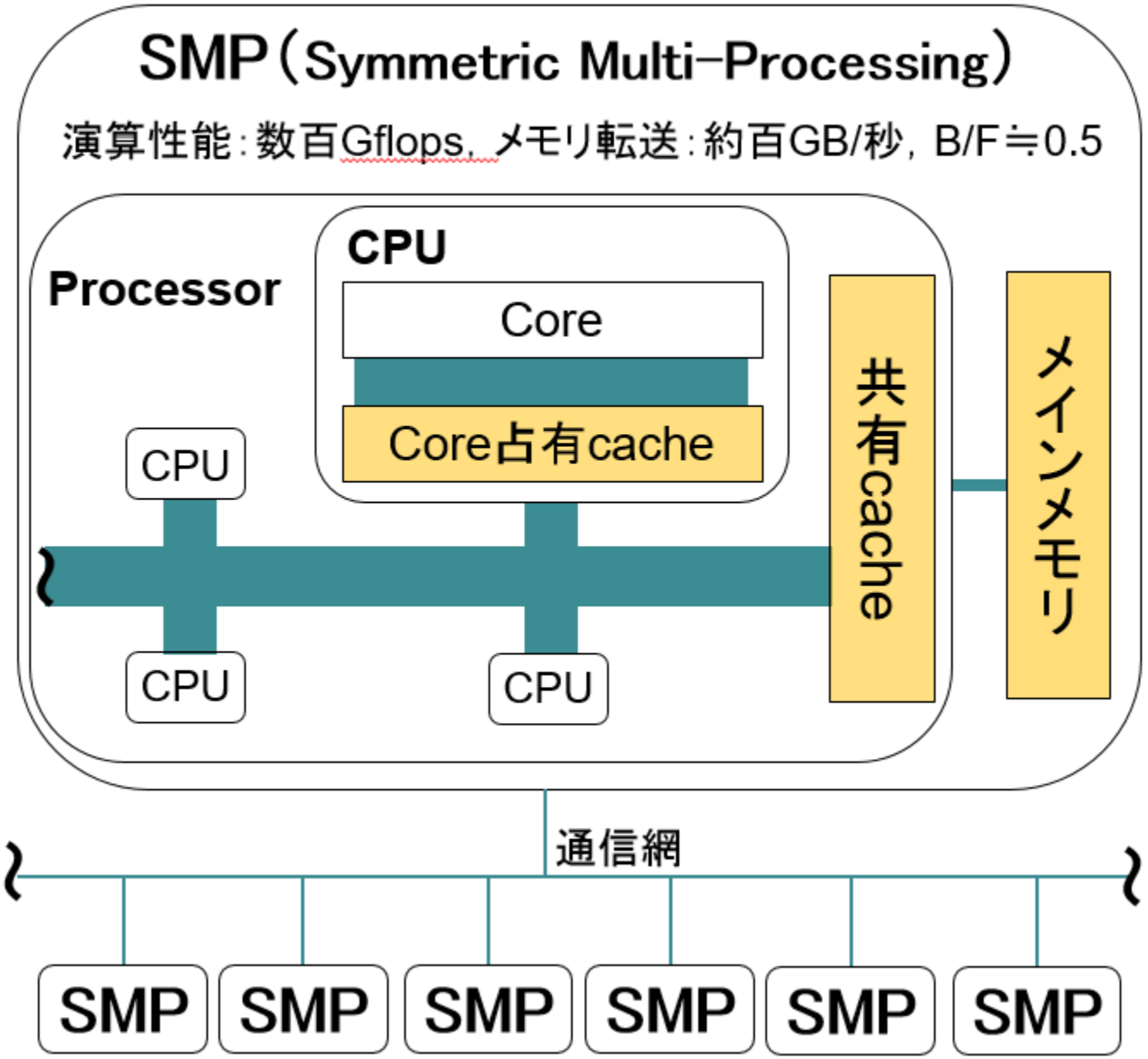
スパコンと線形計算ライブラリ（BLAS, LAPACK）

東京大学情報基盤センター　特任助教　河合 直聡

チュートリアルの流れ

1. 現代のスパコンと並列計算
2. BLAS, LAPACKの説明
3. プログラム実習 I
   - BALS/DGEMMサンプルプログラム実行
4. プログラム実習 II
   - LAPACK/DGESVサンプルプログラム実行

SMPクラスタ型並列計算システムの概念図



■転送速度

Core占有cache

∨ 約10倍

共有cache

∨ 約10倍

メインメモリ

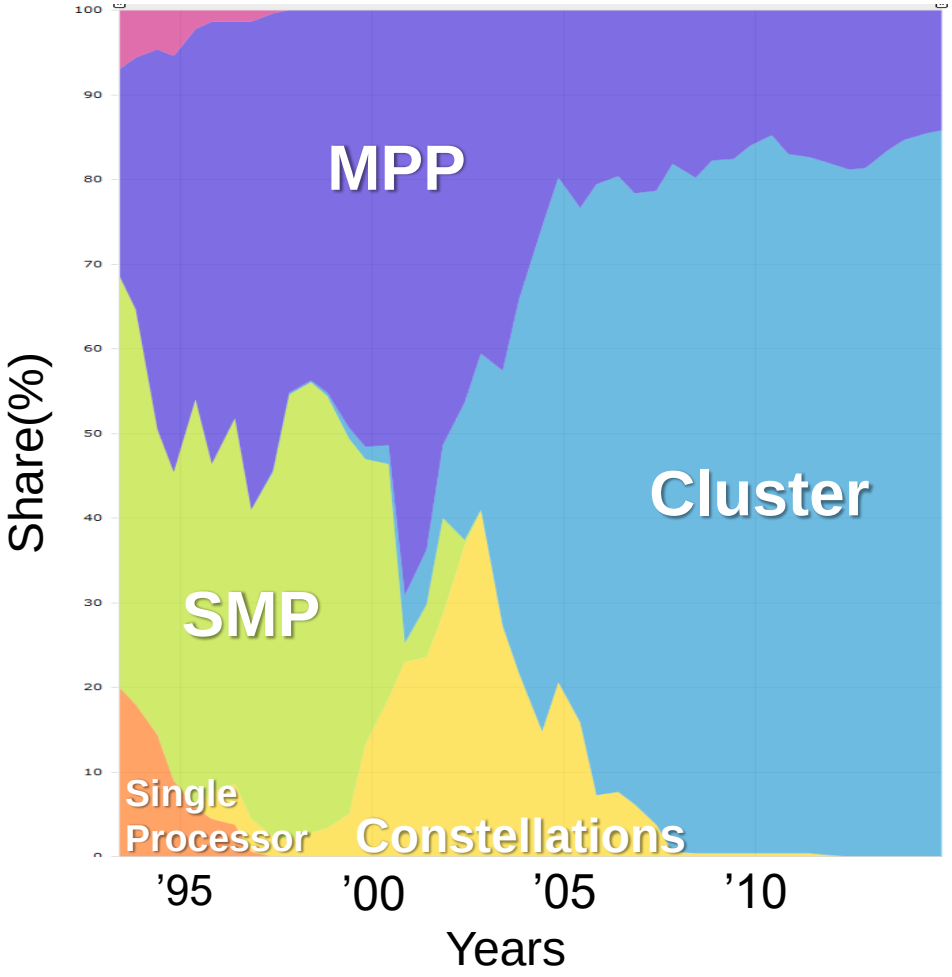
∨ 約10倍

通信網

■容量

- Core占有cache：
  数十～数百KB
- 共有cache：
  数十MB
- メインメモリ：
  数十GB

スーパーコンピュータの変遷



Top 500 list of Supercomputers (http://www.top500.org)

スーパーコンピュータのアーキテクチャの変遷

■クラスタ型が現代では主流

■クラスタ型計算システムの性能を発揮させるのは大変

- キャッシュの有効利用
- SIMDの活用
- 計算負荷バランス
- データ競合回避
- 計算順序保障
- ハイパースレッドの活用
- 通信量の削減
- 効率的な通信パターン
- 同期回数の削減

⇨ ライブラリを活用することで
プログラミングコスト削減

SMPクラスタ型並列計算システムの例

SMPクラスタ名		Wisteria/BDEC01 Odyssey ＠東大基盤センタ	Oakbridge-CX ＠東大基盤センタ
全体	SMP数	7,680	1,368
	SMP間 データ転送性能	56Gbit/s	100Gbit/s
SMP	Processor数	1	2
	理論ピーク演算性能	25.9 Pflops	4.8 PFlops
	メモリ容量	32GB HBM2	192GB (DDR4)
	メモリ転送速度	1,024GB/sec (MCDRAM)	281.6 GB/sec
Processor	Processor名	Fujitsu A64FX	Intel Xeon Platinum 8280 TM
	Core数	48	28
	理論演算性能(Core)	3.38TFlops	86.4 GFlops
	Core辺りcache(L2)	32MB	25MB
	共有cache(L3)	なし	38.5MB

■GB:Giga Byteの略, 1GB=8Gbit

■flops : 1秒間に行われる浮動小数点演算の回数
計算機の性能を表す指標によく使用される

並列計算の必要性と種類

■逐次コードで使える計算資源は、スパコン全体の極々一部
- スパコンは、普通のCPUの集合に過ぎずちっともスーパーではない
- 単体CPUの性能は、普通のパソコンと大差ない

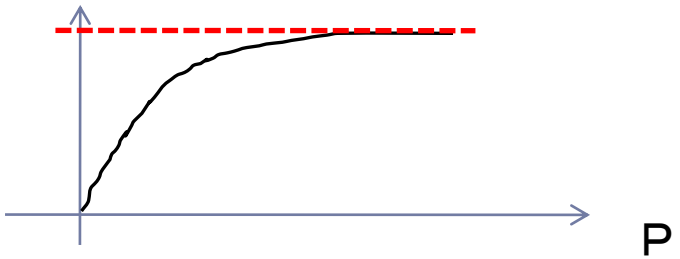
■並列計算の種類

① 逐次コードの単純並列実行
- 小規模計算を大量のパラメータに対して行う
  - ✓ Xcryptが効率的にこれを行えるツール
- 何も工夫することなく、最高の並列計算効率
- アルゴリズムの工夫やCacheの使い方などが高速化の鍵

② 並列コード
- 複数のCPUを連携させて、大規模計算を行う
- 並列化効率を向上させるため、うまく連携させる工夫が必要
- Cacheの使い方など逐次コードで鍵となった技術もやはり重要

性能評価指標－台数効果

- 台数効果
  - 式: $S_P = T_S / T_P$
  - T_S :逐次の実行時間、 T_P :P台での実行時間
  - $S_P = P$ のとき、理想的な(ideal)速度向上
- 並列化効率
  - 式: $E_P = S_P / P \times 100$ [%]
- 飽和性能
  - 速度向上の限界
  - Saturation、「さちる」



アムダールの法則

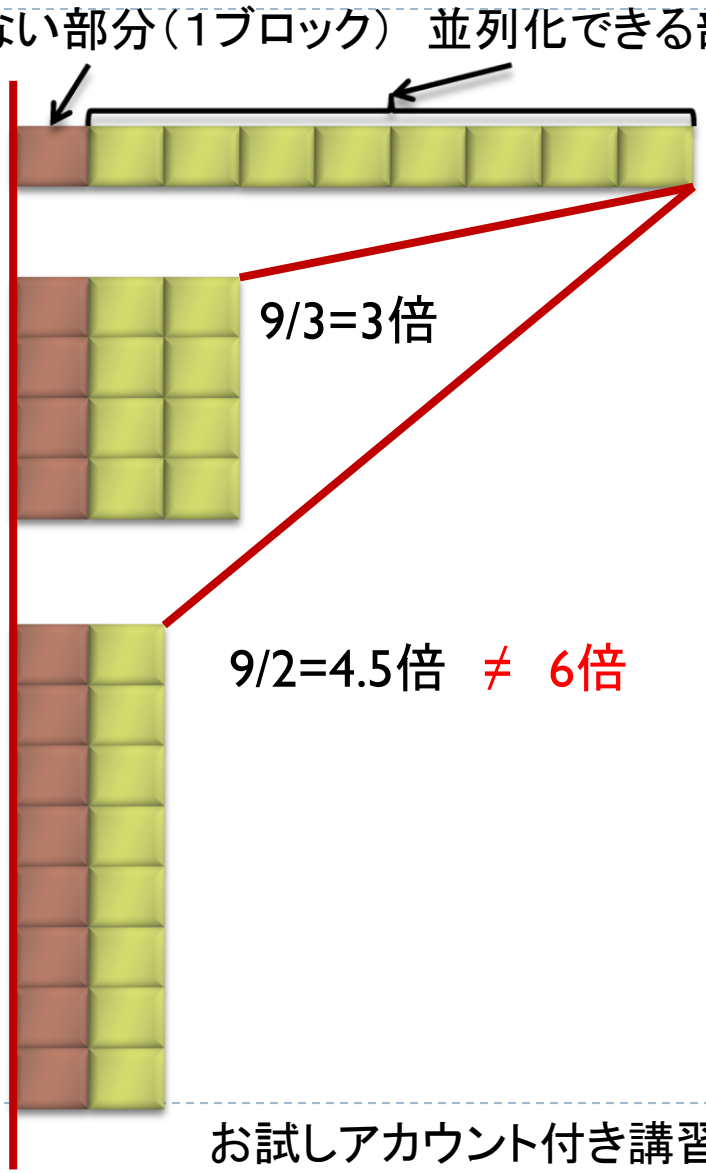
- 逐次実行時間を K とする。
  そのうち、並列化ができる割合を α とする。
- このとき、台数効果は以下のようになる。

$$S_P = K/(K\alpha/P + K(1-\alpha)) = 1/(\alpha/P + (1-\alpha))$$

- 上記の式から、たとえ無限大の数のプロセッサを使っても(P→∞)、台数効果は、高々 1／(1－α) である。
  (アムダールの法則)
  - 並列化が全体の90%だと、無限個のプロセッサを使用しても、
    1/(1-0.9) = 10 倍 にしかならない！

  →高性能を達成するためには、少しでも並列化効率を上げる実装をすることがとても重要である

アムダールの法則の直観例



●逐次実行

＝88.8%が並列化可能

●並列実行（4並列）

9/3=3倍

●並列実行（8並列）

9/2=4.5倍 ≠ 6倍

並列コードの種類

MPIプロセスとOpenMPスレッドを用いた並列化が二大メジャー手段

■プロセス並列

- **MPI（Message Passing Interface）**
- HPF（High Performance Fortran）

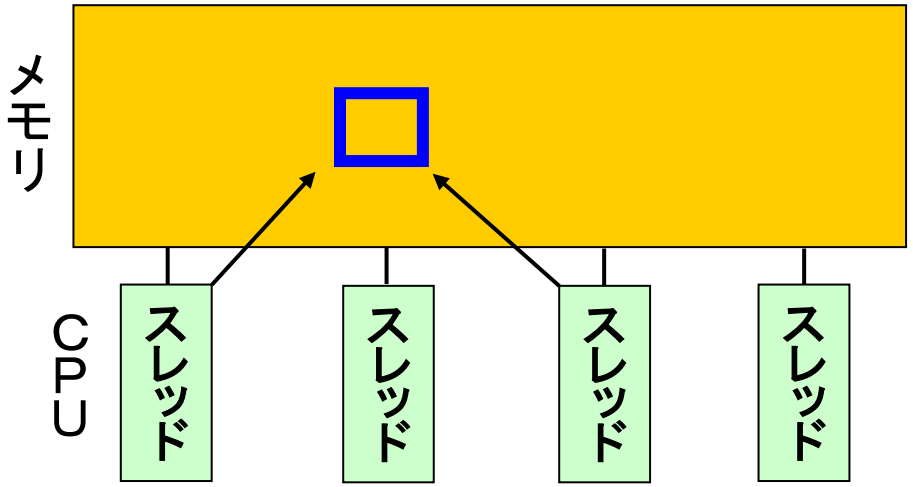
■スレッド並列

- **Open MP**
  - ユーザが並列化指示行を記述
- Open ACC
- Pthread (POSIX スレッド)
- CILK
- Java

プロセスとスレッドの違い
•メモリを意識するかどうかの違い
　•別メモリは「プロセス」
　•同一メモリは「スレッド」

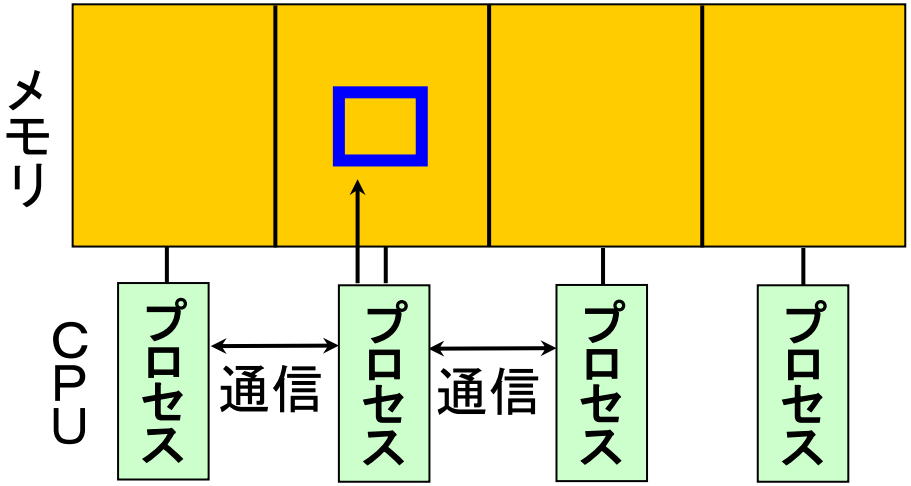
スレッド並列とプロセス並列

■スレッド並列



- SMPのように、メモリが多数のCPUで共有されていることが前提
- 任意の情報へどのスレッドからも直接アクセス可能
- 通信網の先にある他のSMPが持つ情報へはアクセス不能

■プロセス並列



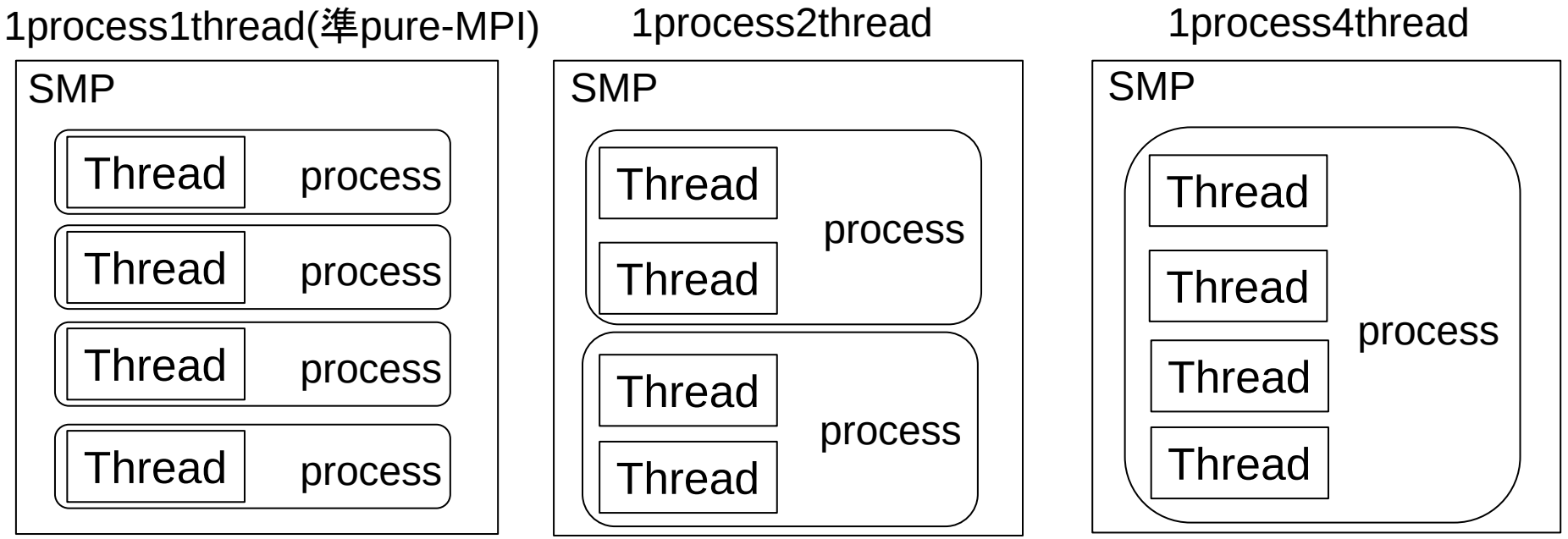
- メモリが多数のCPUで共有されていたとしてもメモリ空間を分離
- 他のプロセスが持つ情報へは「通信」することでアクセス
- 「通信」することにより、他のSMPが持つ情報へもアクセス可能

MPI＋OpenMPハイブリッドプログラミングモデル

■SMPクラスタを考えるとプロセス並列とスレッド並列の組み合わせが合理的

- 各MPI-processが、1つ以上,SMP-core数以下の固定数のOpenMP-Threadを持つ
- 演算の多くはOpenMP-Threadが共有メモリ下で実行する
- 各MPI-processは独立したメモリ空間を有し，process間通信でデータをやりとりする
- 各MPI-processが何Thread持つのが良いかは計算システムと実装で決まる

■SMP-core数が4つの場合の組み合わせ



MPIの概要と特徴

■ メッセージパッシング用のライブラリ規格の1つ

- メッセージパッシングのモデルである
- コンパイラの規格、特定のソフトウエアやライブラリを指すものではない

■ 分散メモリ型並列計算機(クラスタ)に対応可能

- 1プロセッサにおけるメモリサイズやファイルサイズの制約を打破可能
- プロセッサ台数の多い並列システムを用いる実行に向く
  - 1プロセッサ換算で膨大な実行時間の計算を、短時間で処理可能

■ 異なるシステムへの移植が容易

API(Application Programming Interface)の標準化

■ スケーラビリティ、性能が良い

通信処理をユーザが記述することによるアルゴリズムの最適化が可能

OpenMPの概要と特徴

- 共有メモリ型並列計算機用のDirectiveの統一規格
- ディレクティヴ（指示行）の形で利用
  - プログラムを並列に実行するための動作をユーザーが明示
  - 挿入直後のループが並列化される等
  - コンパイラがサポートしていなければ，コメントとみなされる
- 逐次コードからの並列化が比較的容易
  - 指示行（ディレクティヴ）を挿入するだけで手軽に並列化が可能
  - ループ単位など少しずつ段階的に並列化が可能
- OpenMPがMPIより簡単ということはない
  - OpenMP実行環境は，依存関係，衝突，デッドロック，競合条件，結果としてプログラムが誤った実行につながるような問題に関するチェックは要求されていない
  - プログラムが正しく実行されるよう構成するのはユーザーの責任

スパコン上でのライブラリ利用

■ライブラリ・ソフトウェア使えばスパコンの性能を楽に引き出せる

■並列化の各ステージごとに様々なソフトウェアが提供されている

① SMPクラスタ対応パッケージソフト

・格子データや定義ファイルを用意するだけで大規模計算が可能

・ユーザは並列化や通信をあまり意識する必要がない

・OpenFOAM，FrontFlow, ppOpen-APPL/BEMなど

② SMPクラスタ対応ライブラリ、クラスタ対応ライブラリ

・通信機能を持つルーチンを呼ぶことで大規模計算を行う

・ライブラリルーチンが行う計算以外の並列計算はユーザが行う

・MPI，Super-LU，ScaLAPACKなど

③ SMP対応ライブラリ、逐次計算用ライブラリ

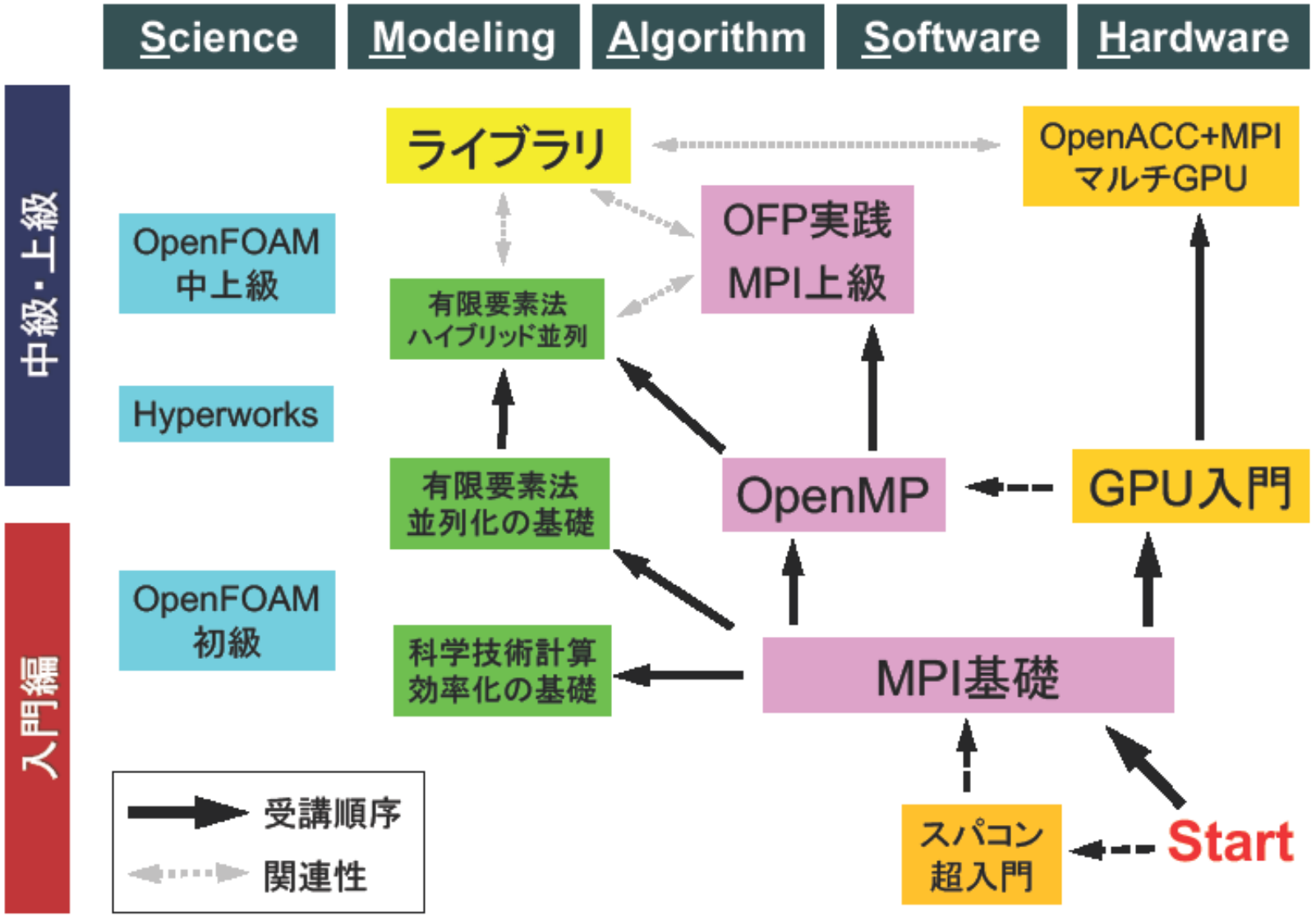
・逐次計算コードで高速化したい部分の機能を有するルーチンを呼ぶ

・複数SMPを使いたければ、通信をユーザが行う必要あり

・Intel MKL，富士通SSL2，BLAS，LAPACKなど

スパコン向け最適化に関する講習会 @ 東大ITC

目的に応じて受講をご検討ください。



BLAS, LAPACK, ScaLAPACKの説明

参考文献（1）

1. BLAS
   http://www.netlib.org/blas/
2. LAPACK
   http://www.netlib.org/lapack/
3. ScaLAPACK
   http://www.netlib.org/scalapack/

Netlib：科学技術計算用ソフトウェア・リポジトリ
（AT&T、ベル研究所、テネシー大学、
オークリッジ国立研究所が共同管理）

BLASとは

- Basic Linear Algebra Subprograms
  （基本線形代数副プログラム集）
  - 線形代数計算で用いられる、基本演算を標準化（API化）したもの。
  - 普通は、密行列の線形代数計算用の基本演算の副プログラムを指す。
  - 疎行列の基本演算用の＜スパースBLAS＞というもののあるが、まだ定着していない。

BLASの意義

- 高性能なライブラリの＜作成の手間＞と、＜プログラム再利用性＞を高める目的で提案
  - BLAS演算の性能改善を、個々のユーザが、個々のプログラムで独立して行うのは、ソフトウエア開発効率が悪い
    - ＜工学的＞でない
  - 性能を改善するチューニングは、経験のないユーザには無理
    - ＜職人芸＞　！＝　＜科学、工学＞
  - BLASは、数学ソフトウエアにおける＜ソフトウエア工学＞のはしり

BLASサブルーチンの分類

- BLASでは、以下のように分類わけをして、サブルーチンの命名規則を統一
  1. 演算対象のベクトルや行列の型（整数型、実数型、複素型）
  2. 行列形状（対称行列、三重対角行列）
  3. データ格納形式（帯行列を二次元に圧縮）
  4. 演算結果が何か（行列、ベクトル）
- 演算性能から、以下の3つに演算を分類
  - レベル1 BLAS: ベクトルとベクトルの演算
  - レベル2 BLAS: 行列とベクトルの演算
  - レベル3 BLAS: 行列と行列の演算
- **なるべく高いレベルのルーチンを使用するのが効果的**

レベル１　BLAS

- ベクトル内積、ベクトル定数倍の加算、など
  - 例：　$y \leftarrow \alpha x + y$
- データの読み出し回数、演算回数がほぼ同じ
- データの再利用（キャッシュに乗ったデータの再利用によるデータアクセス時間の短縮）がほとんどできない
  - 使用による性能向上が、あまり期待できない
  - ほとんど、計算機ハードウエアの演算性能
- レベル1BLASのみで演算を実装すると、演算が本来持っているデータ再利用性がなくなる
  - 例：行列-ベクトル積を、レベル1BLASで実装

レベル２ BLAS

- 行列-ベクトル積などの演算
  - 例：$y \leftarrow \alpha A x + \beta y$
- 前進/後退代入演算、$T x = y$（Tは三角行列）をxについて解く演算、を含む
- レベル1BLASのみの実装よる、データ再利用性の喪失を回避する目的で提案
  - データアクセス時間を、実装法により短縮
  - レベル1BLASに比べ性能を出しやすい（が十分でない）

レベル３ BLAS

- 行列-行列積などの演算
  - 例： $C \leftarrow \alpha A B + \beta C$
- レベル２　BLASより更にデータ再利用を追及
- 行列-行列積では、行列データ　$O(n^2)$ に対して
  演算は $O(n^3)$ なので、データ再利用性が原理的に高い。
- 行列積は、アルゴリズムレベルでもブロック化できる。
  さらにデータの局所性を高めることができる。

BLASの命名規則

- 関数名：XYYYY
  - X：　データ型
    S:単精度、D：倍精度、C：複素、Z：倍精度複素
  - YYYY：　計算の種類
    - レベル1：
      例：AXPY：ベクトルをスカラー倍して加算
    - レベル2：
      例：GEMV: 一般行列とベクトルの積
    - レベル3：
      例：GEMM:一般行列どうしの積

BLASルーチン一覧

LEVEL1 BLAS

zrotg	zdcal	drotg	drot	drotm	zdrot	zswap
dswap	zdscal	dscal	zcopy	dcopy	zaxpy	daxpy
ddot	zdotc	zdotu	dznrm2	dnrm2	dasum	izasum
idamax	dzabs1					

LEVEL2 BLAS

zgemv	dgemv	zgbmv	dgbmv	zhemv	zhbmv	zhpmv	dsymv
dsbmv	ztrmv	zgemv	dgemv	zgbmv	dgemv	zhemv	zhbmv
zhpmv	dsymv	dsbmv	dspmv	ztrmv	dtrmv	ztbmv	ztpmv
dtpmv	ztrsv	dtrsv	ztbsv	dtbsv	ztpsv	dger	zgeru
zgerc	zher	zhpr	zher2	zhpr2	dsyr	dspr	dsyr2
dspr2							

LEVEL3 BLAS

zgemm	dgemm	zsymm	dsymm	zhemm	zsyrk	dsyrk	zherk
zsyr2k	dsyr2k	zher2k	ztrmm	dtrmm	ztrsm	dtrsm	

インタフェース例：DGEMM　（1 / 4）

- **DGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)**
  - C := alpha*op(A)*op(B) + beta*C　の計算をする
  - op(X) = X　もしくは　op(X) = X'（Xの転置行列）
- 引数
  - TRANSA（入力）- CHARACTER*1
    - TRANSA は op(A) の操作を指定する。以下の文字列を指定。
      - TRANSA = 'N' もしくは 'n', op(A) = A
      - TRANSA = 'T' もしくは 't', op(A) = A'
      - TRANSA = 'C' or 'c', op(A) = A'
  - TRANSB（入力）- CHARACTER*1
    - TRANSB は op(B) の操作を指定する。以下同様。

インタフェース例：DGEMM　（2 / 4）

- M（入力）- INTEGER
  - op(A) と　行列 Cの行の大きさを指定する。
- N（入力）- INTEGER
  - op(B) と　行列 Cの列の大きさを指定する。
- K（入力）- INTEGER
  - op(A) の列の大きさ、および op(B) の行の大きさを指定する。
- ALPHA（入力）- DOUBLE PRECISION
  - スカラ値　ALPHAの値を設定する。
- A（入力）- DOUBLE PRECISION
  - 行列Aの配列。大きさは (LDA, ka)で、ka はTRANSA = ‘N’ or ‘n’のときは k。そうでないときは、m。
  - TRANSA = ‘N’ or ‘n’のときは、m × kの配列の要素に行列Aを含まないといけない。そうでないときは、k × mの配列に行列Aの転置を入れる。

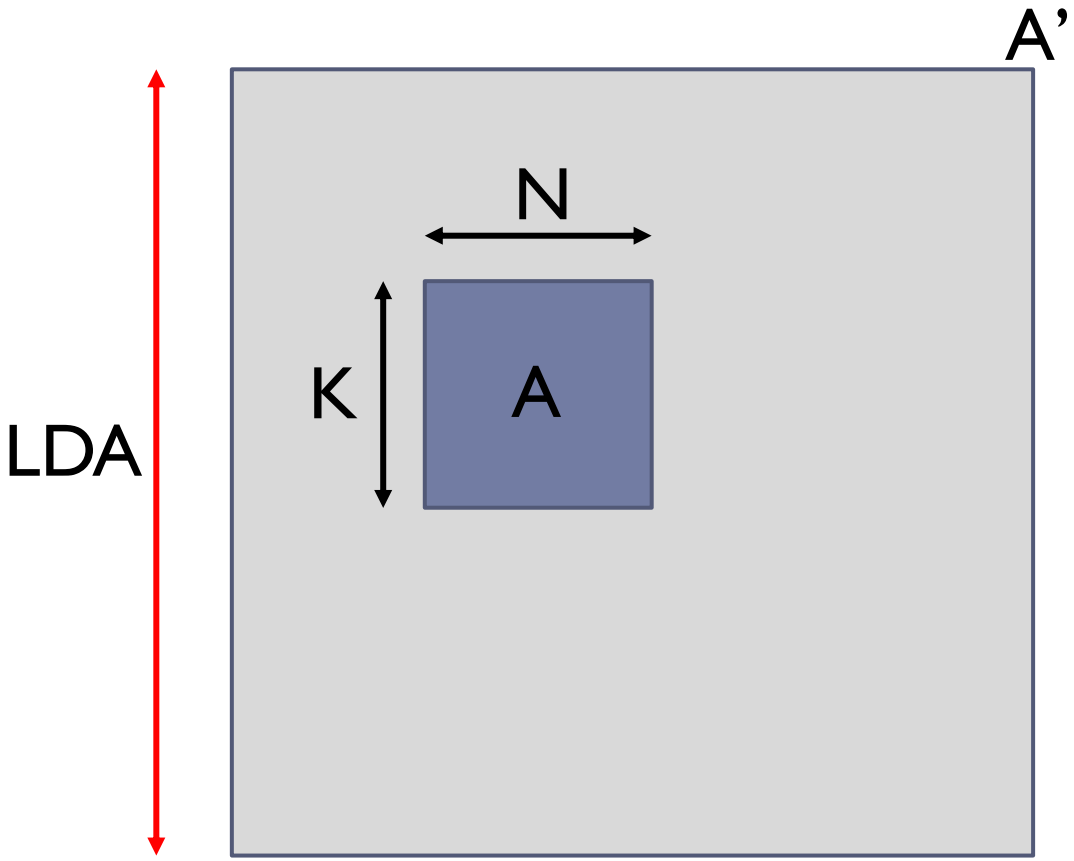
インタフェース例：DGEMM　（3 / 4）

- LDA（入力）- INTEGER
  - 行列Aの最初の次元数を入れる。TRANSA = ‘N’ もしくは ‘n’ なら、LDA は　max(1, m)でなくてはならない。そうでないなら、LDA はmax(1, k)でなくてはならない。
- B（入力）- DOUBLE PRECISION
  - 行列Bの配列。大きさは (LDB, kb)で、kb はTRANSA = ‘N’ or ‘n’のときは n。そうでないときは、k。
  - TRANSA = ‘N’ or ‘n’のときは、k×nの配列の要素に行列Bを含まないといけない。そうでないときは、n×kの配列に行列Bの転置を入れる。
- LDB（入力）- INTEGER
  - 行列Bの最初の次元数を入れる。TRANSA = ‘N’ もしくは ‘n’ なら、LDA は　max(1, k)でなくてはならない。そうでないなら、LDB はmax(1, n)でなくてはならない。

インタフェース例：DGEMM　（4 / 4）

- BETA（入力）- DOUBLE PRECISION
  - スカラ値　BETAの値を設定する。
- C（入力／出力）- DOUBLE PRECISION
  - 行列Cの配列。
  - 入力時、m × n の配列に行列Cを入れる。配列には、BETAが0でない限り、行列Cを入れる。この場合は、Cの入力は必要ない。
  - 出力時、この配列に、m × n行列の演算結果
    (alpha*op(A)*op(B) + beta*C)が上書きされて戻る。
- LDC（入力）- INTEGER
  - 行列Cの最初の次元数を入れる。LDC は　max(1, m)でなくてはならない。

BLASのLDAについて



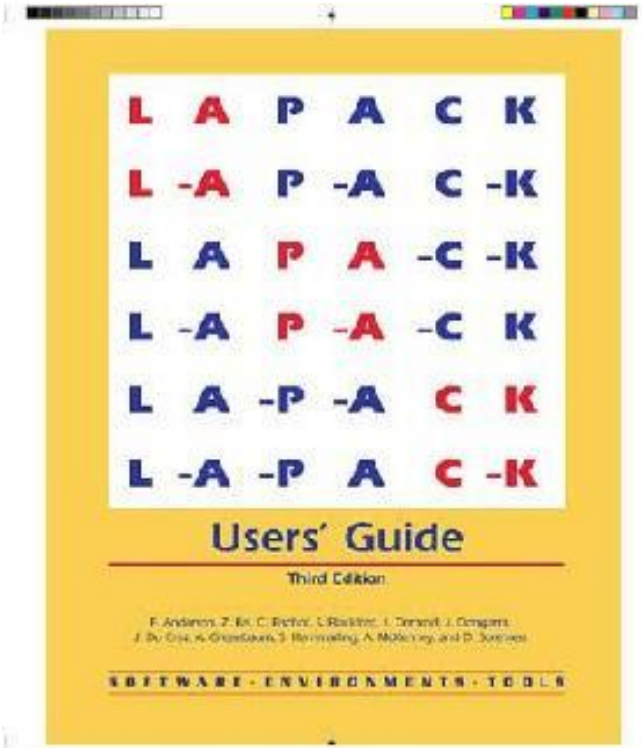
Fortran:
A(1,2)のアドレス
＝A(1,1) + LDA

ＢＬＡＳの問題点

- BLASの問題点
  1. BLASを用いると、データ再利用性や並列性が低下するかもしれない
     - 例：レベル1BLASを用いた行列・ベクトル積
  2. インタフェースに合わせるため、無駄な処理（配列への代入等）が必要になる場合も
     - ＜メモリ浪費＞や＜演算性能低下＞の要因に
  3. ソースコードが読みにくくなる
     - BLASのインタフェースを熟知しないと、かえって処理が理解できない

LAPACKとは

- Linear Algebra PACKage
  （線形代数パッケージ）
  - BLASを組み合わせて使用することで、より高度な線形代数計算を実現
  - 線形方程式系の解法
  - 固有値問題の解法
  - 特異値計算
  - 最小二乗法
- 使いたいものを探すのが大変なほど数多くのルーチンが用意されている
- HP：http://www.netlib.org/lapack/



LAPACKの命名規則

- 関数名：XYYZZZ
  - X：　データ型
    S:単精度、D：倍精度、C：複素、Z：倍精度複素
  - YY：　行列の型
    BD: 二重対角、DI：対角、GB：一般帯行列、GE：一般行列、HE:複素エルミート、HP：複素エルミート圧縮形式、SY：対称行列、….
  - ZZZ：　計算の種類
    TRF:　行列の分解、TRS：行列の分解を使う、CON：条件数の計算、RFS：計算解の誤差範囲を計算、TRI：三重対角行列の分解、EQU：スケーリングの計算、…

LAPACKルーチン一覧

■先頭にデータ型(“s”,“d”,”c”,”z”)のいづれかを付けるとルーチン名になる

bdsdc bdsqr disna gbbrd gbcon gbequ gbequb gbrfs gbrfsx gbsv gbsvx gbsvxx gbtrf gbtrs gebak gebal gebrd gecon geequ geequb gees geesx geev geevx gehrd gejsv gelqf gels gelsd gelss gelsy geqlf geqp3 geqpf geqrf geqrfp gerfs gerfsx gerqf gesdd gesv gesvd gesvj gesvx gesvxx getrf getri getrs ggbak ggbal gges ggesx ggev ggevx ggglm gghrd gglse ggqrf ggrqf ggsvd ggsvp gtcon gtrfs gtsv gtsvx gttrf gttrs hgeqz hsein hseqr opgtr opmtr orgbr orghr orglq orgql orgqr orgrq orgtr ormbr ormhr ormlq ormql ormqr ormrq ormrz ormtr pbcon pbequ pbrfs pbstf pbsv pbsvx pbtrf pbtrs pftrf pftri pftrs pocon poequ poequb porfs porfsx posv posvx posvxx potrf potri potrs ppcon ppequ pprfs ppsv ppsvx pptrf pptri pptrs pstrf ptcon pteqr ptrfs ptsv ptsvx pttrf pttrs sbev sbevd sbevx sbgst sbgv sbgvd sbgvx sbtrd sfrk spcon spev spevd spevx spgst spgv spgvd spgvx sprfs spsv spsvx sptrd sptrf sptri sptrs stebz stedc stegr stein stemr steqr sterf stev stevd stevr stevx sycon syequb syev syevd syevr syevx sygst sygv sygvd sygvx syrfs syrfsx sysv sysvx sysvxx sytrd sytrf sytri sytrs tbcon tbrfs tbtrs tfsm tftri tfttp tfttr tgevc tgexc tgsen tgsja tgsna tgsyl tpcon tprfs tptri tptrs tpttf tpttr trcon trevc trexc trrfs trsen trsna trsyl trtri trtrs trttf trttp tzrzf

インタフェース例：DGESV　（1 / 3）

- **DGESV**
  (N, NRHS, A, LDA, IPIVOT, B, LDB, INFO)
  - *A X = B* の解の行列*X*を計算をする
  - *A * X = B*、ここで *A* は*N ×N*行列で、*X* と *B* は *N ×NRHS*行列とする。
  - 行交換の部分枢軸選択付きのLU分解 で*A* を *A = P * L * U* と分解する。ここで、*P* は交換行列、*L* は下三角行列、*U*は上三角行列である。
  - 分解された*A* は、連立一次方程式*A * X = B*を解くのに使われる。
- 引数
  - N (入力) - INTEGER
    - 線形方程式の数。行列Aの次元数。 N >= 0。

インタフェース例：DGESV　（２／３）

- NRHS (入力)　– INTEGER
  - 右辺ベクトルの数。行列Bの次元数。NRHS >= 0。
- A (入力／出力) – DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、N×Nの行列Aの係数を入れる。
  - 出力時は、Aから分解された行列LとU　= P*L*Uを圧縮して出力する。Lの対角要素は1であるので、収納されていない。
- LDA (入力) – INTEGER
  - 配列Aの最初の次元の大きさ。LDA　>= max(1,N)。
- IPIVOT (出力)　– DOUBLE PRECISION, DIMENSION(:)
  - 交換行列Aを構成する枢軸のインデックス。行列のi行がIPIVOT(i)行と交換されている。

インタフェース例：DGESV　（３／３）

- B (入力／出力) – DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、右辺ベクトルの　N×NRHS　行列Bを入れる。
  - 出力時は、もし、INFO = 0　なら、N×NRHS行列である解行列Xが戻る。
- LDB (入力)　ーINTEGER
  - 配列Bの最初の次元の大きさ。LDB >= max(1,N)。
- INFO (出力) ーINTEGER
  - = 0: 正常終了
  - < 0: もし INFO = -i　なら i-th 行の引数の値がおかしい。
  - > 0: もし INFO = i　なら U(i,i) が厳密に0である。分解は終わるが、Uの分解は特異なため、解は計算されない。

LAPACK95 (紹介のみ)

- LAPACKのFortran95インターフェイス
  - LAPACKが77で書かれているので、90/95以上からの利便性を上げるために利用
    - Fortran90/95用モジュール
    - 関数呼び出しの簡略化（引数の削減）
    - 関数の総称手続き化
  - 本学スパコンには未インストール（各自でのインストールは可）
- HP： http://www.netlib.org/lapack95/

LAPACKの例

```
call DGESV (N,  NRHS,  A, LDA, IPIVOT,  B, LDB, INFO)
```

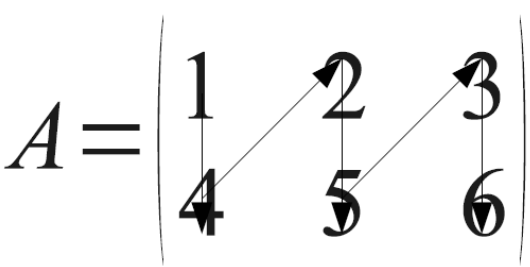
LAPACK95の例

```
use f95_lapack
call LA_GESV (A, B) (型依存なし)
```

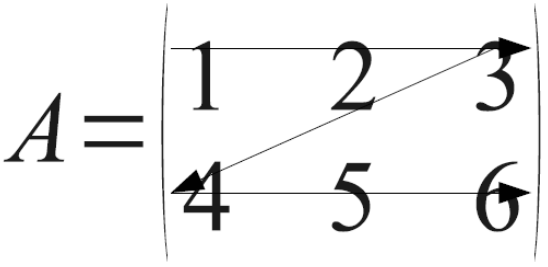
C言語でＢＬＡＳ，LAPACKを使う時の注意

■BLAS, LAPACKはFortranで書かれている

- ルーチン名が変えれている（DGEMM⇒dgemm_）
- 引数はポインタ引き渡し
- 配列インデックスは1-オリジン
- ２次元配列はColomnMajor



ColomnMajor
Fortran



Row Major
C言語

- 複素数は（基本的に）扱えない（後述）

CBLAS、CLAPACK (紹介のみ)

- CBLAS
  - BLASのC用Interface
    - RowMajor, Column Majorを選択可能
    - 複素数は構造体で扱う
    - BLASのページにて提供
    - HP : http://www.netlib.org/blas/#_cblas
- CLAPACK
  - LAPACKをCで完全に書き換えたライブラリ
    - f2cで自動変換
    - Column Majorのみ
    - 複素数は構造体で扱える
    - HP : https://www.netlib.org/clapack/

プログラム実習Ⅰ (BLAS/DGEMM)

サンプルプログラムの説明（BLAS）

- 1コア（逐次）実行版です
- スレッド並列化には、スレッド並列化版をリンクしてコンパイルの上、並列実行数の指定をする必要があります
  - BLASスレッドでは、以下のどちらかの方法でスレッド数を指定します。
    - ジョブスクリプト内に使用スレッド情報を記載

      #PJM --omp thread=10
    - 実行ファイルの前に環境変数で指定

      export OMP_NUM_THREADS=10

      or export MKL_NUM_THREADS=10

実習を行うあたっての注意点(BLAS DGEMM)

- サンプルプログラムのファイル名
  Mat-Mat-BLAS-OBCX.tar
- ジョブスクリプトファイルmat-mat-blas.bash　中のキュー名を、
  lecture から tutorial に変更してから
  pjsub　してください
  - lecture : 実習時間外のキュー（同時実行数1）
  - tutorial : 実習時間内のキュー（同時実行数4+）

BLAS DGEMMのサンプルプログラムの実行（C言語版/Fortran言語版共通）

- $ cd /work/gt00/tXXXXX (XXXXXは数値)
  // 実行してWork領域に移動する
- 以下のコマンドを実行する
  $ cp /work/gt00/z30113/tut193/Mat-Mat-BLAS-OBCX.tar ./
  $ tar xvf Mat-Mat-BLAS-OBCX.tar
  $ cd Mat-Mat-BLAS
  $ cd C //C言語の人
  $ cd F //Fortran言語の人
  $ make
  $ pjsub mat-mat-blas.bash
- 実行が終了したら、以下を実行する
  $ cat mat-mat-blas.bash.oXXXXXX (XXXXXXは数値)

BLAS DGEMMのサンプルプログラムの実行（C言語版）

- 以下のような結果が見えれば成功

N = 9100

Mat-Mat time = 190.748147 [sec.]

7901.214364 [MFLOPS]

OK!

BLAS DGEMMのサンプルプログラムの実行（Fortran言語版）

- 以下のような結果が見えれば成功

```
N  =         9100
Mat-Mat time[sec.] =     103.923363756388
MFLOPS =     14502.4366198134
 OK!
```

サンプルプログラムの説明（C言語）

- ./mat-mat-blas 9100
  のように、実行時引数で行列サイズを指定できます。
- #define DEBUG 1
  「1」にすると、行列-行列積の演算結果が検証できます。
- MyMatMat関数の仕様
  - Double型N×N行列AとBの行列積をおこない、Double型N×N行列Cにその結果が入ります

時間計測方法（C言語）

```
double  t0, t1, t2, t_w;
..
ierr = MPI_Barrier(MPI_COMM_WORLD);
t1 = MPI_Wtime();

<ここに測定したいプログラムを書く>

ierr = MPI_Barrier(MPI_COMM_WORLD);
t2 = MPI_Wtime();

t0 =  t2 - t1;
ierr = MPI_Reduce(&t0, &t_w, 1,
       MPI_DOUBLE,MPI_MAX, 0,
       MPI_COMM_WORLD);
```

バリア同期後
時間を習得し保存

各プロセッサーで、t0の値は異なる。
この場合は、最も遅いものの値をプロセッサ0番が受け取る

時間計測方法（Fortran言語）

```
  double precision t0, t1, t2, t_w
  double precision MPI_WTIME
..
  call MPI_BARRIER(MPI_COMM_WORLD, ierr)
  t1 = MPI_WTIME(ierr)

  ＜ここに測定したいプログラムを書く＞

  call MPI_BARRIER(MPI_COMM_WORLD, ierr)
  t2 = MPI_WTIME(ierr)

  t0 =  t2 - t1
  call  MPI_REDUCE(t0, t_w, 1,
&       MPI_DOUBLE_PRECISION,
&       MPI_MAX, 0, MPI_COMM_WORLD, ierr)
```

バリア同期後
時間を習得し保存

各プロセッサーで、t0の値は異なる。
この場合は、最も遅いものの値をプロセッサ0番が受け取る

OBCXでのBLAS呼び出し

- インテルコンパイラから、BLAS (Intel MKLライブラリ)を呼び出す場合、以下のオプションを付けます。

●BLASのリンク(並列実行対応)
mpiifort -mkl <プログラム名> :Fortran言語
mpiicc -mkl <プログラム名> :C言語

●逐次版を明示的にリンクする場合
mpiifort –mkl=sequential <プログラム名> : Fortran言語
mpiicc –mkl=sequential <プログラム名> : C言語

演習課題１（BLAS）

1. MyMatMat関数の行列積コードを、BLAS のDEGMM ルーチンを呼び出して高速化してください
   - コードで行列-行列積の実行部をコメントアウト
     C版：ファイルmat-mat-blas.cの106行目
     Fortran版：ファイルmat-mat-blas.fの117行目
   - DEGMMの引数の並びに注意して記述
     C版：ファイルmat-mat-blas.cの115行目
     Fortran版：ファイルmat-mat-blas.fの126行目
   - C言語は、以下に注意してください
     - 関数名： DGEMM_
     - 全ての引数が ポインタ引き渡し

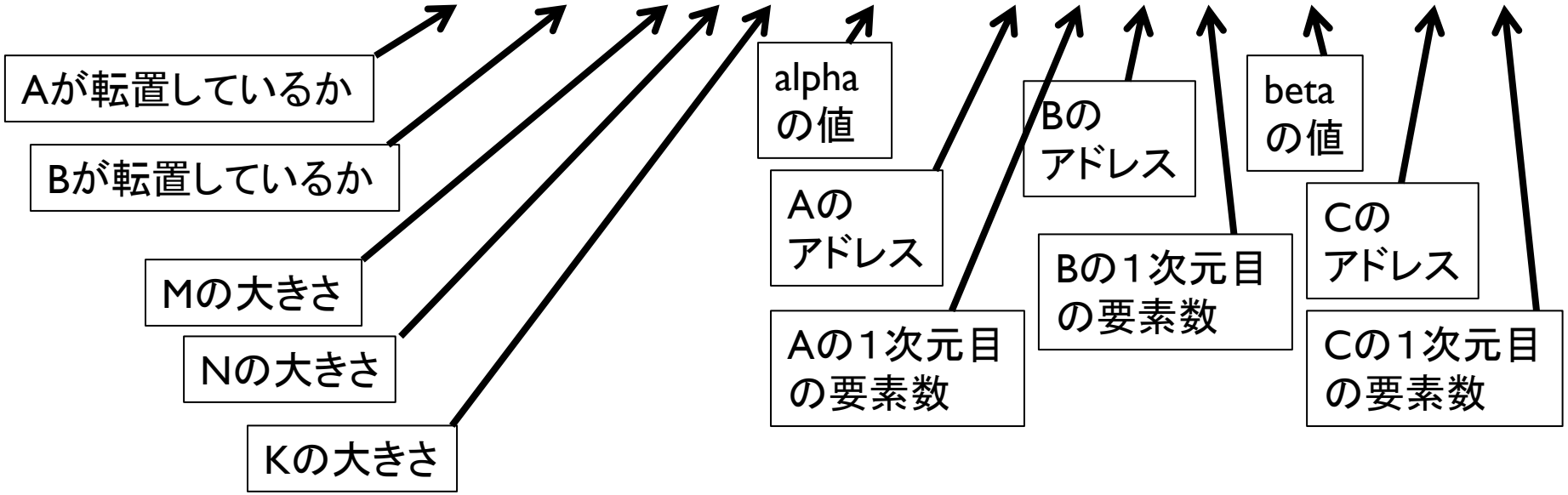
演習課題１のヒント

- 倍精度演算BLAS3

  C := alpha*op(A)*op(B) + beta*C

  A: M*K;　B:K*N;　C:M*N;

  CALL DGEMM(‘N’, ‘N’, n, n, n, ALPHA, A, N, B, N, BETA, C, N)



演習課題２ （BLAS）

2-1 スレッド並列コードの実行

スレッド並列版のDGEMMを呼び出し、
いろいろなスレッド数で実行してください

ヒント1：3ページ前のスライドを見て、
ファイル「Makefile」を開き、「FLAGS=」
の右辺を書き換える

ヒント2：ファイル「mat-mat-blas.bash」を開き、
「thread=1」の右辺を変更してジョブを投入する

2-2 台数効果による性能向上の評価

逐次実行に比べ、どれだけ高速化されたか、
性能評価をしてください

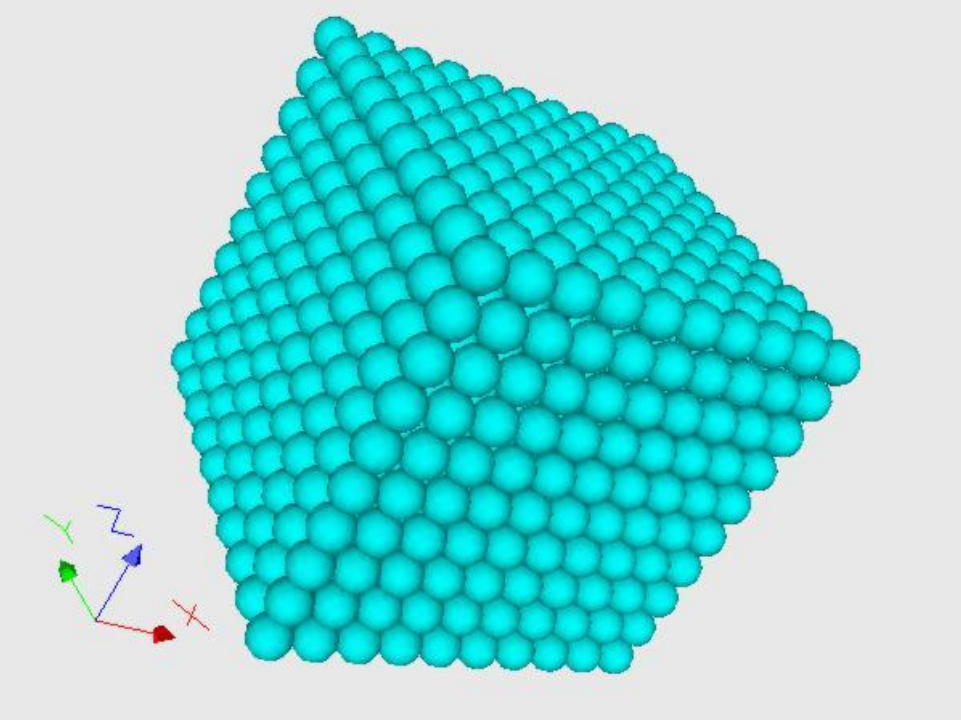
プログラム実習Ⅱ (LAPACK)

内容

- サンプルプログラムの説明
- LAPACK/DGESVサンプルプログラム実行（C言語、Fortran言語）

サンプルプログラムで計算する問題

■格子状に並んだ粒子間の熱伝導問題



■詳細資料

http://nkl.cc.u-tokyo.ac.jp/FEMintro/

サンプルプログラム・シリアル版の計算手順

- 制御データ入力
- 粒子生成
- 係数マトリクス計算
  - 熱伝導
  - 対流熱伝達
  - 発熱
- CG法による連立一次方程式求解（test_org.f, test_org.c）
  - 密行列
  - 点ヤコビ前処理
- 出力
  - MicroAVS用

サンプルプログラムの説明

- 誤差ERRの計算は，10×10×10の問題に特化されています．
  それ以外の問題サイズでは機能しません．
- test_org.c、test_org.fは、ライブラリを用いず、手製の反復解法（CG法）を用いて、線形方程式系の求解を行うコードです。

サンプルプログラムの実行（LAPACK/DGESV）

サンプルプログラムの説明（ LAPACK版）

- 1コア（逐次）実行版です
- スレッド並列化版は、スレッド並列化版をリンクしてコンパイルの上、並列実行数の指定をする必要があります。
- LAPACKはスレッド並列化のみ対応しているため、複数ノードを通信しつつ利用することはできません
  - ノード内のみLAPACKを使い、ノード間はMPIで並列化するような使い方は可能です
    - たとえば、領域分割法を利用し、ノード内のみLAPACKで連立一次方程式を解く場合

実習を行うあたっての注意点(LAPACK dgesv)

- サンプルプログラムのファイル名
  lecLAPACK-OBCX.tar
- ジョブスクリプトファイルgo.sh中のキュー名を
  lecture から tutorial に変更してから
  pjsub してください。
  - lecture : 実習時間外のキュー（同時実行数1）
  - tutorial : 実習時間内のキュー（同時実行数4+）

LAPACK dgesvのサンプルプログラムの実行（C言語版/Fortran言語版共通）

- /work/gt00/t00***に移動する
- 以下のコマンドを実行する

```
$ cp /work/gt00/z30113 /tut193 /lecLAPACK-OBCX.tar ./
$ tar xvf lecLAPACK-OBCX.tar
$ cd sphere-LAPACK
  $ cd C //C言語の人
  $ cd F //Fortran言語の人
$ make
$ pjsub go.bash
```

- 実行が終了したら、以下を実行する

```
$ cat go.bash.oXXXXXX (XXXXXXは数字)
```

LAPACK dgesvのサンプルプログラムの実行（C言語版）

- 以下のような結果が見えれば成功

```
time  = 0.0000001 [sec.]
 13083407467.39 [MFLOPS]
990 -2.272792e+00
991 -2.276715e+00
992 -2.288410e+00
993 -2.307670e+00
994 -2.334166e+00
995 -2.367479e+00
996 -2.407125e+00
997 -2.452584e+00
998 -2.503330e+00
999 -2.558846e+00
err = 6.274235e+02
```

連立一次方程式求解部分が
実装されて無いので、
時間が極端に短く、
誤差ERRが大きく
なっていますが、
正常な動作です。

LAPACK dgesvのサンプルプログラムの実行（Fortran言語版）

- 以下のような結果が見えれば成功

```
TIME[sec] =  6.05359673500061OE-008
MFLOPS =   11070575549.3297
   991   -2.272792E+00
   992   -2.276715E+00
   993   -2.288410E+00
   994   -2.307670E+00
   995   -2.334166E+00
   996   -2.367479E+00
   997   -2.407125E+00
   998   -2.452584E+00
   999   -2.503330E+00
  1000   -2.558846E+00
ERR =   627.423511391423
```

連立一次方程式求解部分が
実装されて無いので、
時間が極端に短く、
誤差ERRが大きく
なっていますが、
正常な動作です。

OBCXでのLAPACK呼び出し

- インテルコンパイラから、LAPACK (Intel MKLライブラリ)を呼び出す場合、コンパイラに以下のオプションを付けます

●LAPACKのリンク(並列実行対応)
mpiifort -mkl <プログラム名> :Fortran言語
mpiicc -mkl <プログラム名> :C言語

●逐次版を明示的にリンクする場合
mpiifort –mkl=sequential <プログラム名> : Fortran言語
mpiicc –mkl=sequential <プログラム名> : C言語

演習課題（LAPACK）

1. test.c もしくは test.f のメイン関数（手続き）の
   連立一次方程式の求解を行う部分
   （test.c:159行目、test.f : 114行目）に、
   LAPACK dgesvルーチンを呼び出してください

- 行列 : ATAN, サイズ : N, 右辺ベクトル : RHS, ピボット:PIV,
- 引数の並びに注意してください
- C言語は、dgesv のFortran手続きの呼び出しになります。以下に注意してください
  - 関数名 : **dgesv_**
  - 全ての引数がポインタ引き渡し
- 誤差（ERR）が十分小さいことを確認してください。

BLAS DGEMM回答

BLAS DGEMM演習問題１の回答（C言語版）

```
double ALPHA, BETA;
char TEX[1] = {'N'};

ALPHA=1.0;
BETA=1.0;
dgemm_(&TEX, &TEX, &n, &n, &n, &ALPHA,
       A, &n,  B, &n, &BETA, C, &n);
```

BLAS DGEMM演習問題１の回答（Fotran言語版）

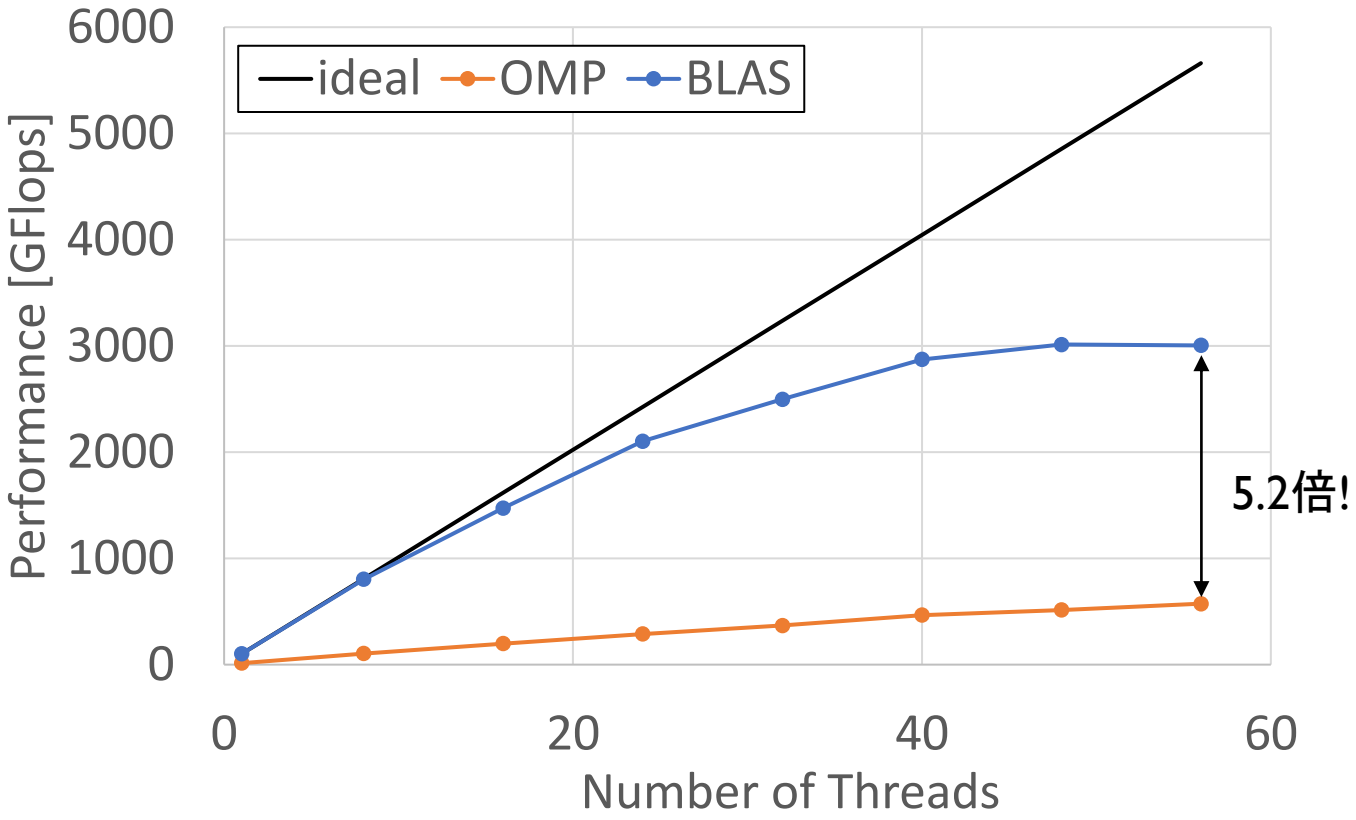
```
      double precision ALPHA,BETA


     ALPHA=1.0d0
     BETA=1.0d0
     CALL DGEMM('N', 'N', n, n, n, ALPHA,
    &           A, n, B, n, BETA, C, n)
```

OpenMP 参考例

Fortran サンプル

```
!$OMP parallel do private(i, j, k)
do i=1, N
  do j=1, N
    do k=1, N
      C(i, j) = C(i, j) + A(i, k) * B(k, j)
    enddo
  enddo
enddo
!$OMP end parallel do
```

性能評価



	性能 [GFlops]		Ratio
	1スレッド	56スレッド	
OMP	14.19	573.16	40.40
BLAS	101.12	3005.07	29.72

演習課題1 LAPACK dgesv回答

演習課題 LAPCK dgesvの回答（C言語版）

```
dgesv_(&nn, &inc, amat2, &nn,
    piv, rhs, &nn, &info);
```

演習課題 LAPCK dgesvの回答（Fotran言語版）

```
      call DGESV(N, INC, AMAT, N,
     &           PIV, RHS, N, INFO)
```

アンケートへ回答のお願い

- 第193回講習会「科学技術計算の効率化入門」アンケートQRコード

https://forms.office.com/r/vsPUeRpNAf

お疲れさまでした